Universidad del Valle de Guatemala - UVG

Facultad de Ingeniería - Computación
Curso: CC3104 - Aprendizaje por Refuerzo **Sección:** 10
Laboratorio 8: Deep Reinforcement Learnign

Autores:

- Diego Alexander Hernández Silvestre - **21270**
- Linda Inés Jiménez Vides - **21169**
- Mario Antonio Guerra Morales - **21008**

In [1]:
```
%pip install torch
%pip install gymnasium
```

```
Looking in indexes: https://pypi.org/simple, https://pypi.ngc.nvidia.comNote: you may need to restart the kernel to use updated packages.

Requirement already satisfied: torch in c:\users\lijv1.linda_hp\anaconda3\lib\site-packages (2.7.0)
Requirement already satisfied: filelock in c:\users\lijv1.linda_hp\anaconda3\lib\site-packages (from torch) (3.13.1)
Requirement already satisfied: typing-extensions>=4.10.0 in c:\users\lijv1.linda_hp\anaconda3\lib\site-packages (from torch) (4.11.0)
Requirement already satisfied: sympy>=1.13.3 in c:\users\lijv1.linda_hp\anaconda3\lib\site-packages (from torch) (1.14.0)
Requirement already satisfied: networkx in c:\users\lijv1.linda_hp\anaconda3\lib\site-packages (from torch) (3.3)
Requirement already satisfied: jinja2 in c:\users\lijv1.linda_hp\anaconda3\lib\site-packages (from torch) (3.1.4)
Requirement already satisfied: fsspec in c:\users\lijv1.linda_hp\anaconda3\lib\site-packages (from torch) (2024.6.1)
Requirement already satisfied: setuptools in c:\users\lijv1.linda_hp\anaconda3\lib\site-packages (from torch) (75.1.0)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in c:\users\lijv1.linda_hp\anaconda3\lib\site-packages (from sympy>=1.13.3->torch) (1.3.0)
Requirement already satisfied: MarkupSafe>=2.0 in c:\users\lijv1.linda_hp\anaconda3\lib\site-packages (from jinja2->torch) (2.1.3)
Looking in indexes: https://pypi.org/simple, https://pypi.ngc.nvidia.com
Requirement already satisfied: gymnasium in c:\users\lijv1.linda_hp\anaconda3\lib\site-packages (1.2.1)
Requirement already satisfied: numpy>=1.21.0 in c:\users\lijv1.linda_hp\anaconda3\lib\site-packages (from gymnasium) (1.26.4)
Requirement already satisfied: cloudpickle>=1.2.0 in c:\users\lijv1.linda_hp\anaconda3\lib\site-packages (from gymnasium) (3.0.0)
Requirement already satisfied: typing-extensions>=4.3.0 in c:\users\lijv1.linda_hp\anaconda3\lib\site-packages (from gymnasium) (4.11.0)
Requirement already satisfied: farama-notifications>=0.0.1 in c:\users\lijv1.linda_hp\anaconda3\lib\site-packages (from gymnasium) (0.0.4)
Note: you may need to restart the kernel to use updated packages.
```

In [2]:
```python
# === Setup & Hiperparámetros (Inciso 4) ===
import math
import random
import collections
import numpy as np
import torch
import torch.nn as nn
import torch.optim as optim
import gymnasium as gym
from collections import deque
import numpy as np
import time
import matplotlib.pyplot as plt
```

In [3]:
```python
SEED = 42
random.seed(SEED)
np.random.seed(SEED)
torch.manual_seed(SEED)

# Hiperparámetros Inciso 4
ENV_ID = "CartPole-v1"

# DQN
GAMMA = 0.99              # tasa de descuento
LR = 0.0001               # tasa de aprendizaje
BATCH_SIZE = 64
BUFFER_CAPACITY = 10_000  # tamaño del replay buffer
TARGET_UPDATE_FREQ = 1000 # pasos para copiar pesos a la target network

# Entrenamiento
MAX_EPISODES = 500
MAX_STEPS_PER_EPISODE = 1000

# Exploración ε-greedy
EPS_START = 1.0
EPS_END = 0.01
EPS_DECAY_STEPS = 20_000
```

In [ ]:
```python
# === Ambiente de Gymnasium (Inciso 1) ===
env = gym.make(ENV_ID)
env.reset(seed=SEED)

obs_dim = env.observation_space.shape[0]
n_actions = env.action_space.n

print(f"Ambiente: {ENV_ID}")
print(f"Dimensión del estado: {obs_dim}")
print(f"Número de acciones: {n_actions}")
```

```
Ambiente: CartPole-v1
Dimensión del estado: 4
Número de acciones: 2
```

In [ ]:
```python
# === Red DQN (Inciso 2) y Consideraciones técnicas (Inciso 3) ===
class DQN(nn.Module):
    def __init__(self, input_dim: int, output_dim: int, hidden: int = 128):
        super().__init__()
        self.net = nn.Sequential(
            nn.Linear(input_dim, hidden),
            nn.ReLU(),
            nn.Linear(hidden, hidden),
            nn.ReLU(),
            nn.Linear(hidden, output_dim),
        )

    def forward(self, x):
        return self.net(x)

policy_net = DQN(obs_dim, n_actions)
target_net = DQN(obs_dim, n_actions)
target_net.load_state_dict(policy_net.state_dict())
target_net.eval()

optimizer = optim.Adam(policy_net.parameters(), lr=LR)
mse_loss = nn.MSELoss()

def epsilon_by_step(step: int):
    frac = min(1.0, step / EPS_DECAY_STEPS)
    return EPS_START + (EPS_END - EPS_START) * frac

def select_action(state: np.ndarray, global_step: int):
    eps = epsilon_by_step(global_step)
    if random.random() < eps:
        return env.action_space.sample(), eps
    with torch.no_grad():
        state_t = torch.tensor(state, dtype=torch.float32).unsqueeze(0)
        q_values = policy_net(state_t)
        action = int(torch.argmax(q_values, dim=1).item())
        return action, eps
```

In [6]:
```python
# === Replay Buffer (Experience Replay) — Inciso 2 ===
Transition = collections.namedtuple(
    "Transition", field_names=["state", "action", "reward", "next_state", "done"]
)

class ReplayBuffer:
    def __init__(self, capacity: int):
        self.capacity = capacity
        self.buffer = collections.deque(maxlen=capacity)

    def push(self, state, action, reward, next_state, done):
        self.buffer.append(Transition(state, action, reward, next_state, done))

    def sample(self, batch_size: int):
        batch = random.sample(self.buffer, batch_size)
        states = torch.tensor(np.array([t.state for t in batch]), dtype=torch.float32)
        actions = torch.tensor([t.action for t in batch], dtype=torch.int64).unsqueeze(-1)
        rewards = torch.tensor([t.reward for t in batch], dtype=torch.float32).unsqueeze(-1)
        next_states = torch.tensor(np.array([t.next_state for t in batch]), dtype=torch.float32)
        dones = torch.tensor([t.done for t in batch], dtype=torch.float32).unsqueeze(-1)
        return states, actions, rewards, next_states, dones

    def __len__(self):
        return len(self.buffer)

replay_buffer = ReplayBuffer(BUFFER_CAPACITY)
print("ReplayBuffer listo con capacidad:", BUFFER_CAPACITY)
```

```
ReplayBuffer listo con capacidad: 10000
```

In [7]:
```python
# === Update step del DQN usando la Target Network — Inciso 2 ===
def dqn_update():
    if len(replay_buffer) < BATCH_SIZE:
        return 0.0

    states, actions, rewards, next_states, dones = replay_buffer.sample(BATCH_SIZE)

    # Q(s,a) actual de la policy_net
    q_values = policy_net(states).gather(1, actions)  # [batch,1]

    # Objetivo con la target_net (sin gradiente)
    with torch.no_grad():
        max_next_q = target_net(next_states).max(dim=1, keepdim=True)[0]
        target = rewards + (1.0 - dones) * GAMMA * max_next_q

    loss = mse_loss(q_values, target)

    optimizer.zero_grad()
    loss.backward()
    torch.nn.utils.clip_grad_norm_(policy_net.parameters(), max_norm=10.0)  # clipping opcional
    optimizer.step()
    return float(loss.item())
```

In [8]:
```python
# Parámetros de entrenamiento
EPISODES_TO_TRAIN = 600
SOLVED_SCORE = 475.0
LOG_EVERY = 10

reward_history = []
avg100_history = []
eps_history = []
loss_history = []
recent_rewards = deque(maxlen=100)

global_step = 0
best_avg100 = -float("inf")
start_time = time.time()

for ep in range(1, EPISODES_TO_TRAIN + 1):
    state, info = env.reset(seed=SEED + ep)
    ep_reward = 0.0
    ep_losses = []

    for t in range(MAX_STEPS_PER_EPISODE):
        action, eps = select_action(state, global_step)
        next_state, reward, terminated, truncated, info = env.step(action)
        done = terminated or truncated

        # Guardar transición en Replay Buffer
        replay_buffer.push(state, action, reward, next_state, done)

        # Avanzar estado y contadores
        state = next_state
        ep_reward += reward
        global_step += 1

        # Actualización del DQN
        loss = dqn_update()
        if loss:
            ep_losses.append(loss)

        # Sincronización periódica de la target network
        if global_step % TARGET_UPDATE_FREQ == 0:
            target_net.load_state_dict(policy_net.state_dict())

        if done:
            break

    reward_history.append(ep_reward)
    recent_rewards.append(ep_reward)
    avg100 = float(np.mean(recent_rewards))
    avg100_history.append(avg100)
    eps_history.append(eps)
    if len(ep_losses) > 0:
        loss_history.append(np.mean(ep_losses))

    if avg100 > best_avg100:
        best_avg100 = avg100

    if ep % LOG_EVERY == 0 or ep == 1:
        print(f"Ep {ep:4d} | R: {ep_reward:6.1f} | ε: {eps:5.3f} | "
              f"avg100: {avg100:6.1f} | buffer: {len(replay_buffer):5d} | step: {global_step}")

    # Criterio de parada temprana (si ya está resuelto)
    if best_avg100 >= SOLVED_SCORE and len(recent_rewards) == recent_rewards.maxlen:
        print(f"\n✅ Ambiente considerado resuelto: avg100={best_avg100:.1f} (≥ {SOLVED_SCORE}) en ep={ep}.")
        break

elapsed = time.time() - start_time
print(f"\nEntrenamiento finalizado en {ep} episodios, tiempo: {elapsed:.1f}s. "
      f"Mejor avg100: {best_avg100:.1f}")
```

```
Ep    1 | R:   18.0 | ε: 0.999 | avg100:   18.0 | buffer:    18 | step: 18
Ep   10 | R:   11.0 | ε: 0.989 | avg100:   23.1 | buffer:   231 | step: 231
Ep   20 | R:   33.0 | ε: 0.977 | avg100:   22.9 | buffer:   458 | step: 458
Ep   30 | R:   14.0 | ε: 0.967 | avg100:   22.5 | buffer:   675 | step: 675
Ep   40 | R:   19.0 | ε: 0.957 | avg100:   21.9 | buffer:   878 | step: 878
Ep   50 | R:   17.0 | ε: 0.945 | avg100:   22.2 | buffer:  1108 | step: 1108
Ep   60 | R:   18.0 | ε: 0.934 | avg100:   22.4 | buffer:  1341 | step: 1341
Ep   70 | R:   19.0 | ε: 0.923 | avg100:   22.4 | buffer:  1565 | step: 1565
Ep   80 | R:   14.0 | ε: 0.912 | avg100:   22.3 | buffer:  1787 | step: 1787
Ep   90 | R:   18.0 | ε: 0.902 | avg100:   21.9 | buffer:  1975 | step: 1975
Ep  100 | R:   21.0 | ε: 0.888 | avg100:   22.6 | buffer:  2260 | step: 2260
Ep  110 | R:   36.0 | ε: 0.874 | avg100:   23.1 | buffer:  2543 | step: 2543
Ep  120 | R:   28.0 | ε: 0.859 | avg100:   23.9 | buffer:  2846 | step: 2846
Ep  130 | R:   30.0 | ε: 0.847 | avg100:   24.1 | buffer:  3088 | step: 3088
Ep  140 | R:   24.0 | ε: 0.831 | avg100:   25.4 | buffer:  3417 | step: 3417
Ep  150 | R:   29.0 | ε: 0.814 | avg100:   26.6 | buffer:  3768 | step: 3768
Ep  160 | R:   13.0 | ε: 0.799 | avg100:   27.1 | buffer:  4054 | step: 4054
Ep  170 | R:   57.0 | ε: 0.778 | avg100:   29.1 | buffer:  4479 | step: 4479
Ep  180 | R:   14.0 | ε: 0.765 | avg100:   29.7 | buffer:  4754 | step: 4754
Ep  190 | R:   75.0 | ε: 0.745 | avg100:   31.8 | buffer:  5152 | step: 5152
Ep  200 | R:   33.0 | ε: 0.719 | avg100:   34.2 | buffer:  5684 | step: 5684
Ep  210 | R:   55.0 | ε: 0.696 | avg100:   36.0 | buffer:  6140 | step: 6140
Ep  220 | R:   15.0 | ε: 0.674 | avg100:   37.5 | buffer:  6595 | step: 6595
Ep  230 | R:   47.0 | ε: 0.633 | avg100:   43.3 | buffer:  7422 | step: 7422
Ep  240 | R:   48.0 | ε: 0.600 | avg100:   46.5 | buffer:  8072 | step: 8072
Ep  250 | R:  143.0 | ε: 0.554 | avg100:   52.3 | buffer:  9002 | step: 9002
Ep  260 | R:  155.0 | ε: 0.498 | avg100:   60.9 | buffer: 10000 | step: 10148
Ep  270 | R:   81.0 | ε: 0.444 | avg100:   67.5 | buffer: 10000 | step: 11229
Ep  280 | R:  208.0 | ε: 0.354 | avg100:   83.0 | buffer: 10000 | step: 13050
Ep  290 | R:  192.0 | ε: 0.247 | avg100:  100.6 | buffer: 10000 | step: 15208
Ep  300 | R:  233.0 | ε: 0.118 | avg100:  121.3 | buffer: 10000 | step: 17816
Ep  310 | R:  252.0 | ε: 0.010 | avg100:  143.2 | buffer: 10000 | step: 20460
Ep  320 | R:  185.0 | ε: 0.010 | avg100:  160.6 | buffer: 10000 | step: 22653
Ep  330 | R:  234.0 | ε: 0.010 | avg100:  177.6 | buffer: 10000 | step: 25185
Ep  340 | R:  243.0 | ε: 0.010 | avg100:  193.2 | buffer: 10000 | step: 27394
Ep  350 | R:  228.0 | ε: 0.010 | avg100:  208.4 | buffer: 10000 | step: 29842
Ep  360 | R:  201.0 | ε: 0.010 | avg100:  220.8 | buffer: 10000 | step: 32233
Ep  370 | R:  235.0 | ε: 0.010 | avg100:  234.6 | buffer: 10000 | step: 34689
Ep  380 | R:  252.0 | ε: 0.010 | avg100:  238.4 | buffer: 10000 | step: 36889
Ep  390 | R:  230.0 | ε: 0.010 | avg100:  240.1 | buffer: 10000 | step: 39216
Ep  400 | R:  194.0 | ε: 0.010 | avg100:  235.8 | buffer: 10000 | step: 41393
Ep  410 | R:  388.0 | ε: 0.010 | avg100:  235.5 | buffer: 10000 | step: 44006
Ep  420 | R:  225.0 | ε: 0.010 | avg100:  232.5 | buffer: 10000 | step: 45906
Ep  430 | R:  203.0 | ε: 0.010 | avg100:  227.5 | buffer: 10000 | step: 47935
Ep  440 | R:  223.0 | ε: 0.010 | avg100:  225.0 | buffer: 10000 | step: 49896
Ep  450 | R:  199.0 | ε: 0.010 | avg100:  219.3 | buffer: 10000 | step: 51772
Ep  460 | R:  180.0 | ε: 0.010 | avg100:  216.2 | buffer: 10000 | step: 53852
Ep  470 | R:  158.0 | ε: 0.010 | avg100:  209.7 | buffer: 10000 | step: 55658
Ep  480 | R:  182.0 | ε: 0.010 | avg100:  205.3 | buffer: 10000 | step: 57417
Ep  490 | R:  150.0 | ε: 0.010 | avg100:  199.2 | buffer: 10000 | step: 59132
Ep  500 | R:  178.0 | ε: 0.010 | avg100:  194.7 | buffer: 10000 | step: 60862
Ep  510 | R:  159.0 | ε: 0.010 | avg100:  186.0 | buffer: 10000 | step: 62610
Ep  520 | R:  176.0 | ε: 0.010 | avg100:  183.6 | buffer: 10000 | step: 64263
Ep  530 | R:  182.0 | ε: 0.010 | avg100:  180.3 | buffer: 10000 | step: 65962
Ep  540 | R:  144.0 | ε: 0.010 | avg100:  176.3 | buffer: 10000 | step: 67530
Ep  550 | R:  139.0 | ε: 0.010 | avg100:  173.7 | buffer: 10000 | step: 69141
Ep  560 | R:  168.0 | ε: 0.010 | avg100:  169.8 | buffer: 10000 | step: 70834
Ep  570 | R:  147.0 | ε: 0.010 | avg100:  166.8 | buffer: 10000 | step: 72334
Ep  580 | R:  142.0 | ε: 0.010 | avg100:  164.6 | buffer: 10000 | step: 73881
Ep  590 | R:  146.0 | ε: 0.010 | avg100:  162.7 | buffer: 10000 | step: 75405
Ep  600 | R:  143.0 | ε: 0.010 | avg100:  161.0 | buffer: 10000 | step: 76960

Entrenamiento finalizado en 600 episodios, tiempo: 176.9s. Mejor avg100: 240.1
```

In [9]:
```python
plt.figure()
plt.plot(reward_history, label="Recompensa por episodio")
plt.plot(avg100_history, label="Media móvil (últimos 100)")
plt.title("Progreso de entrenamiento DQN")
plt.xlabel("Episodio")
plt.ylabel("Recompensa")
plt.legend()
plt.show()
```